**Spring Core/ Spring MVC and Spring Boot**

**Framework is used to build the enterprise application , with more concentration on business logic instead of boilerplate code for security, logging,**

**Spring Core**

* **Spring framework creates and manages objects.**
* **Spring does not depend on anything like tomcat. Has its own IOC container**
* **Spring read the config file and read all the class from the config file, create/store objects for each class in IOC container.**
* **Spring Bean - Objects which are created and managed by spring in IOC container is known as Spring bean. Bean means object in IOC container**
* **getBean(String id); -- will give u the object of class, id is mentioned in bean class**

**1.IOC Container (Spring IOC – Spring Inversion of Control)**

1. **It is used to create/manage our object s**
2. **Helping our application to be configurable**
3. **Managing/ injecting dependencies**
4. **Two types of IOC container:**

**a. BeanFactory - it has basic functionalities. It is Interface, it is Legacy**

**b. ApplicationContext - it provides all the features**  **that provides BeanFactory and additional features also. It is Interface, it is advanced.**

* **ClassPathXmlApplicationContext class implements ApplicationContext. This class has all the required methods.**

**Eg program:**

**Class Mobile {**

**Public static void main (String[] args){**

**ApplicationContext context = new classPathXmlApplicationContext(“beans.xml”);**

**//it will create IOC container object. It will look beans.xml in class path (src folder)**

**//beans.xml is configuration class**

**}**

**}**

**Bean.xml**

**<beans ….>**

**<bean id=”airtel” class=”package/Airtel”></bean>**

**</beans>**

**2. Dependency Injection**

**Why DI is required:**

* **Class employee has address dependency , if we create obj with new key word like Address adr = new Address(41/1, 4t street,namakkal)**
* **Now employee class depend on address class. So if I change anything in Address class employee class also impact.**
* **If I am going to add one more field in Address class we need to change the object creation in all the classes wherever we created address obj.**
* **To overcome this dependency issue we are using dependency injection Design pattern**

**What is DI:**

* **To create object for class A, make sure all the dependencies of the class has been initialized.**
* **To initialize the values for our dependencies, usually we will hard code the values in class A, like username, Object creation and initialization.**
* **Dependency Injection simply means, injecting our dependency. We should have some values in our dependency those values are injected by using Spring**
* **Instead of creating obj for dependencies inside the class and injecting the dependencies**

**Eg:**

**Class Me{**

**String name; //Dependencies in form of Literals**

**Int homeNo; // Dependencies in form of Literals**

**Family f; // Dependencies in form of Objective**

**Job j; // Dependencies in form of Objective**

**ArrayList<Integer > impNos; // Dependencies in form of Collection**

**// all these variables are I'm depending. those values need to be injected.**

**// here spring is going to inject the values to the dependencies**

**}**

* **Spring is injecting the values using 2 ways for Literals**

1. **Using Setter injection**
2. **Using Constructor Injection**

**1.Using Setter injection**

* **In general, if we want to initialize the value for literal, we will use setter method to set the values. Which is called Setter injection.**
* **Eg: we are having Student class, and we need to initialize name property/literal of student class then we will use below code**

**p.s.v.m(){**

**Student s – new Student ();**

**s.setName(“Maya”);**

**}**

* **But in Spring, Spring Inversion of Control (Spring IOC) managing our dependencies (4 th point).**
* **Create student bean in bean.xml, now spring ioc will create object.**
* **Create application context object and get student object from IOC.**
* **Now we have only student object. Name Is not injected. Because to inject dependency of literals in class we need to update bean tag in bean.xml with property tag with name and initial value to inject.**
* **Spring using setter method to inject values when we use property tag in bean.xml**

**2. Using Constructor Injection**

* **Injecting the property values using constructor Is known as constructor injection.**
* **Constructor is used to initialize non static properties of class during the time of object creation.**
* **In general, we will create object like Student stu = new Student (1,” Maya”); to create object and initialize id,name property of student class using constructor. But again, it is kind of manually creating object**
* **Instead of this we can ask spring to inject those values by using bean.xml’s below bean tag,**

**<bean id=” student” class”package/classname”>**

**<constructor-arg name=”studentName” value=”Maya”>**

**<constructor-arg name=”id” value=”1” type=”int”>**

**</bean>**

**Inject Object Dependencies:**

* **Usually we will use Mathcheat m = new MathCheat();**
* **But using new keyword is tight coupling. hence, we have to avoid new keyword.**
* **Instead of that we have to create object for ‘m’ in other class like in main class create object for m and using setter method set the object and after that call any method of m.**

**Student s = new student ();**

**MathCheat cheat = new MathCheat(); // it is dependency of student class**

**s.setMathCheat(cheat); // injecting math cheat object using setter method**

**s.cheating(); // calling cheating method which again will call method of math cheat class**

* **But this above step can be done by spring by adding below things in bean.xml,**

**<bean id=” student” name=” package/className”> // created student class object internally**

**<property name=”mathCheat”>**

**<bean name=” package/className”></bean> // created math cheat object internally and using setter methods it is injecting.**

**</property>**

**</bean>**

* **Here if we are going to create one more student object the mathcheat object also will create one more time.**
* **It makes heavy weight , to make it light weighted we have to below below things**

**<bean id=”mathCheatDemo” name=” package/className”></bean> // created math cheat object internally and using setter methods it is injecting.**

**<bean id=” student” name=” package/className”> // created student class object internally**

**<property name=”mathCheat” ref=” mathCheatDemo”/>**

**</bean>**

* + **In this case only one mathcheat object will be created and it is shared to all the student object.**

**Loose Coupling:**

* **Interface help us to achieve loose coupling in java**
* **Inside student class use interface reference and remove implementation class reference, and add the class reference bean.xml based on our requirements.**

**3) Autowired**

* **Autowired is used to inject dependency automatically**
* **@Autowire annotation is applicable only for object type not String, primitive type.**
* **We can inject object by using autowire property of bean tag.**
* **Here property is autowire=”byName”. Here by name Is name of the dependent class ex: heart**

**<bean id=”heart” class””></bean>**

**<bean id=”human” class=”packge/class” autowire=”byName”></bean>**

* **Here we already created object to heart using bean (id should be heart. It is important)**
* **While creating human class object any dependency present in Human class meeting autowire=”byName” criteria, inject those beans to their respective dependency.**
* **If it is not possible to create bean id same as dependency name we can use byType like below.**

**<bean id=”heartObject” class=” package/heart”></bean>**

**<bean id=”human” class=”package/human” autowire=”byType”></bean>**

* **While creating human class obbject any dependency present in Human class meeting autowire=”byType” criteria(in beans.xml file), inject those beans to their respective dependency.**
* **Here heart variable type in human class and heartObject bean type(class name) in bean.xml are matched.**
* **We can inject dependency using constructor also.**

**<bean id=”heartObject” class=” package/heart”></bean>**

**<bean id=”human” class=”package/human” autowire=”constructor”></bean>**

* **Here we need to create constructor in human class and initialize heart value**
* **Instead of configuring autowire property in xml, lets configure auto wiring at java side using spring’s @Autowire annotation.**
* **We can use @autowire in constructor, setter methods, also (by name, by type, constructor)**
* **While configure autowire with constructor we need to create defualt constructor also and add <context:annotation-config/> in bean.xml file.**
* **By name and by type injections achieved by setter methods**
* **Initially byType is resolved, if byType is fails byName will resolve.**
* **In bean.xml if 2 heart objects are there then it will look for byName**
* **If there is no matched name found then throw exception**

**4)@Qualifier**

* **It is the annotation**
* **It is used when in autowire there is 2 heart object created and both id’s are different from name.**
* **Using @ Qualifier we can mention what object we are going to use from bean.xml to the setter method.**
* **Use @Qualifier(“humanHeart”) after @Autowire annotation**
* **We can add this @autowire,@Qualifier before variable name(dependency) also**
* **If we added before dependency no need of writing setter method and it is not required.**
* **Spring will directly create and inject your object to the matched dependency.**

**5.@Value**

* **@Value annotation is used to injecting values from properties file or environmental values or external configuration like cloud config.**
* **@Value(“${student.name: test}”) -- here student.name is property name and test are sample value, if no value given for name property test will take.**
* **In bean.xml we have to activate the annotation by using below tag,**

**<context:annotation-config/>**

* **Whenever we are adding @value, @Autowired (any) annotation before variable/ filed the setter method we don’t need.**

**NOTE: can we apply @value before object?**

**6.@Required**

* **@Required annotation is used to restrict if the particular dependency not injected, so not create object for the class**
* **It is used to make the injection of dependency mandatory for the specific property**
* **If we are used @Required for name property and value not initialized then it throws bean initialization exception.**

**NOTE: can we apply @required before variable/feild?**

**Spring Annotations**

**1.@Component**

* **If spring finds that any class with @Compoenent annotation then will assume this is component so spring will create object for that class and put/register into applicationContext/IOC Container**
* **It is equivalent to <bean> tag in beans.xml**
* **Simply if we annotated class with @Component object will created automatically for that class**
* **We can use @Component(“collegeBean”)-- here collegeBean is bean name. Using bean name, we can get the bean**

**2.@ComponentScan**

* **If we added this annotation with our java class, then it will scan our components in current package(if src )/ specified package and instantiate (create object) for our component class.**
* **@ComponentScan(“package\_name”) in our configuration class**
* **If we are not using @Component anywhere then no need for @ComponentScan also**

**3.@Configuration**

* **This bean.xml is configuration file, where we are doing all the configuration, instead of that we can create configuration class using @Configuration annotation**
* **Using configuration annotation, we can make class as configuration class.**
* **Hence now we are using class we have to change our applicationcontext code like, replace bean.xml name with configuration class name to get bean name.**

**4.@Bean**

* **Similar to @Component this annotation also used to create bean for the class**
* **If our java class not associated with any annotation we can use @Bean to create bean**
* **Inside our configuration class create one method to create the college object and return it.**
* **Here method name will be bean id.**

**Eg:**

**@Configuration**

**public class CollegeConfig**

**{**

**@Bean**

**public College collegeBean()**

**{**

**return new College ();**

**}**

**}**

* **We can give customize the bean name using @Bean (name=” defaultBean”)**
* **We can give multiple alias name using @Bean (name= {“colBean” ,”defaultBean”})**

**Note: what is difference between the @Bean and @Component**

**5.@Autowired**

* **When we use @component, bean for that class has been created**
* **To inject the created bean in required placed we can use @Autowired for class and interface**
* **If we are using @autowired before the field no need to write setter method**

**6.@Value**

* **Using @Value annotation we can inject the value for literals from application properties, environment variable,**
* **If we use @value before field then no need of setter methods**
* **Syntax: @Value(“${property\_name}”)**

**7.@Primary**

* **If the interface has 2 implementation class and both are annotated with @Component, if we are using the @Autowired with interface name in class it will throw exception like, excepted single match bean but found two.**
* **To resolve above issue we can use @Primary in one of implementation class to indicate this is the primary bean I want to inject**
* **If you want to inject non primary object we can use @Qualifier**

**8. @Qualifier**

* **This annotation used while autowired if we found 2 bean we can specify using this @Qualifier(“bean\_id”) we can specify which object I want to inject.**

**9.** @ConfigurationpProperties

* this is the annotation that tells Spring that we have external properties specified in a Java file

**Bean Life Cycle**

**1.@PostConstruct**

* **When we annotate this with any method , that method will be called automatically after creating object for the respective class, we don’t need to call manually.**
* **It will call automatically once object constructed.**
* **It will act as init method**

**2.@PreDestroy**

* **This method will call automatically before destroy the object**
* **We can have the clean up activities in this method.**
* **It will act as destroy method,**

**Spring MVC**

**Model – Source – generating / storing the data (Storage)-**

**View – UI – end user sees it -**

**Controller – mediator b/w model and view (Mechanism) -**

**View ->Controller -> model**

* **When we type something in chrome it will hit controller class (bridge b/w client and server)**
* **Controller is just mediator and don’t hv any data . So it will call storage**
* **Storage has all the data and return those requested by controller**
* **Controller returns the response**
* **View should be in jsp/html etc**
* **Controller is servlet**
* **Model is data base / file system**

**MVC Benefits**

**Note:**

1. **Spring MVC will help you to develop java j2EE (web) applications easily. But the problem is setting up your project (or working environment manually by doing a lot of configurations by using XML etc.)**
2. **Spring boot helps you to wrap all spring components in a convenient way where no external XML configuration is needed.so the spring boot wraps spring MVC itself**
3. **Spring will automatically initialize the class having a @Controller annotation and register that class with the spring container.**

**Dispatcher Servlet:**

* **DispatcherServlet is a default front controller available in spring.**
* **Front controller will accept all the request from client (through common url: /drugdisease.com/\*), and make decisions that which controller class can handle each incoming request and redirect the request to respective controller. Get the response and return the same to client.**
* **It provides the way to centralize all the incoming request to the web application**
* **Add dispatch servlet name and mapping web.xml file, (right click project, and create servlet)**
* **Whenever the dispatcher servlet initialize it is looking for dispatcherServletname-servlet.xml under web-inf folder.**
* **Dispatcherservlet initialized only at the time of first request coming**
* **To initialize the dispatcher servlet, we need to add load on startup tag under in servlet tag in web.xml**
* **Also, dispatcherservlet create webApplication context container which will be created on server start up and closed at the time of server shut down.**
* Spring Boot auto-configures a **Dispatcher Servlet** if **Spring MVC jar** is on the classpath.
* Auto-configures a **Data Source** if **Hibernate jar** is on the classpath.

**In spring boot**

Spring Boot provides the *spring-boot-starter-web* library for developing web applications using Spring MVC. One of the main features of Spring Boot is autoconfiguration. **The Spring Boot autoconfiguration registers and configures the *DispatcherServlet* automatically**. Therefore, we don’t need to register the *DispatcherServlet* manually.

By default, the *spring-boot-starter-web* starter configures *DispatcherServlet* to the URL pattern “/”. So, we don't need to complete any additional configuration for the above *DispatcherServlet* example in the *web.xml* file. However, we can customize the URL pattern using *server.servlet.*\* in the *application.properties* file:

server.servlet.context-path=/demospring.mvc.servlet.path=/baeldung

https://www.baeldung.com/spring-boot-dispatcherservlet-web-xml

**AOP**

* **AOP is aspect-oriented programming**
* **It is a methodology similar to OOP, with which we will create the application.**
* **AOP allows to remove the cross cutting concern from our application and separate it from main business logic.**
* **when we create application, for example shopping cart apps , suppose if we want to logging for each request in application, authentication/authorization these things will be done for all the request , which is not business logic and general functionality.**
* **These cross cutting concern are known as Aspect**
* **we can create logger class for logging and annotate that class with @Aspect**
* **we can add @Before(“execution(\* demo.ShoppingCart.checkout())”) annotation with logger class method . so that before invoking checkout method logger class method will be invoked., similarly we can call logger method after ,before return value,after return value using respective annotation.**

**Application context vs web applcation context**

**Front controller vs controller**